

시스템 프로그래밍

Assignment3-1

Class : A
Professor : 김태석 교수님
Student ID : 2022202065
Name : 박나림

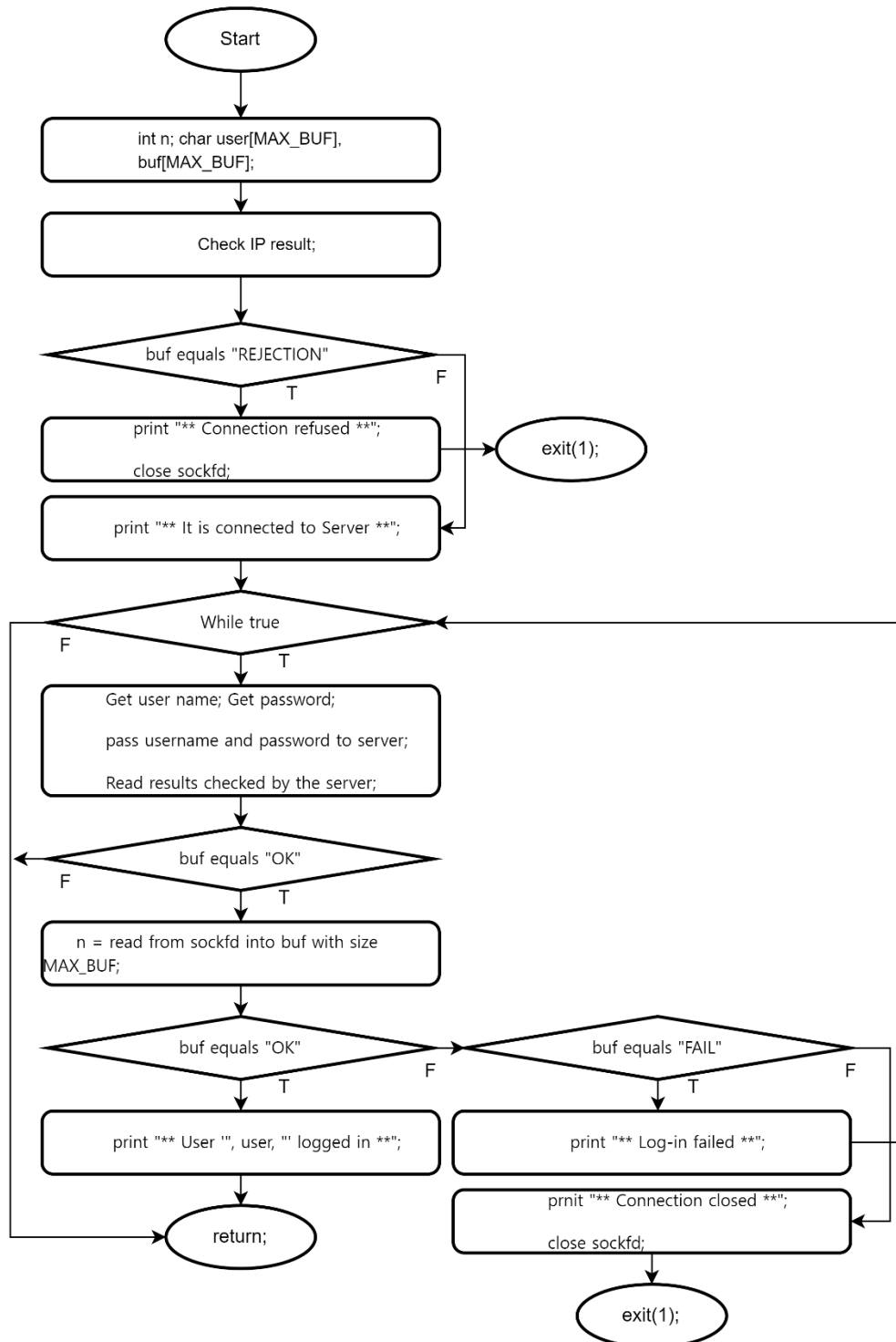
Introduction

FTP 를 구현하는 프로젝트 중 User authentication, access control 를 다루는 과제이다. 클라이언트와 서버가 연결될 때 인증하는 작업을 거치는 과정을 구현하는 것이 목표이다.

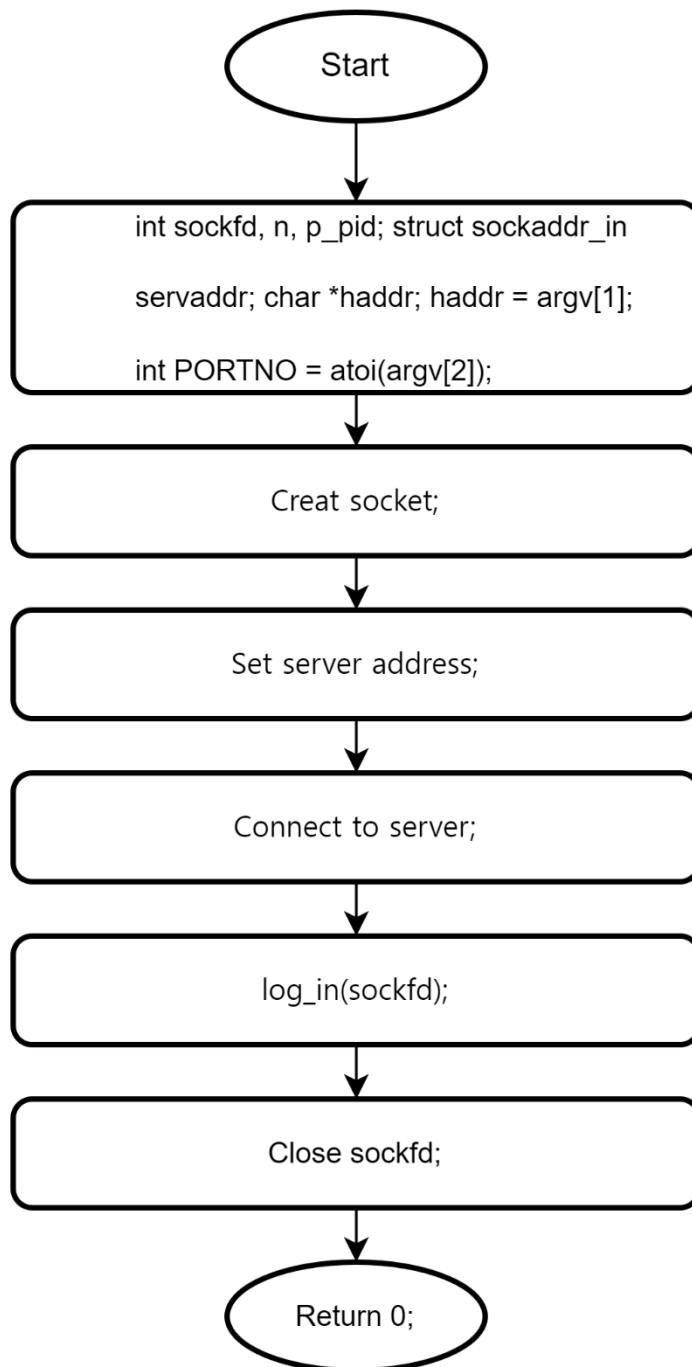
클라이언트와 서버가 소켓으로 연결될 때, 서버는 클라이언트의 IP 가 접속 가능한 IP 주소인지 'access.txt' 파일을 통해 검사한다. 유효한 IP 라면 연결을 하고, 아니라면 연결을 거부한다. 연결에 성공할 시 클라이언트는 서버에게 로그인 시도를 하게 된다. 이때 서버는 전달받은 user name 과 password 가 'passwd' 파일에 저장된 정보인지 검사하게 된다. 검사 결과는 클라이언트에게 전달된다. 성공이라면 종료, 실패라면 클라이언트는 최대 3 회 다시 로그인 시도를 할 수 있다. 3 회 모두 실패 시 서버에서 연결을 종료시킨다.

Flow chart

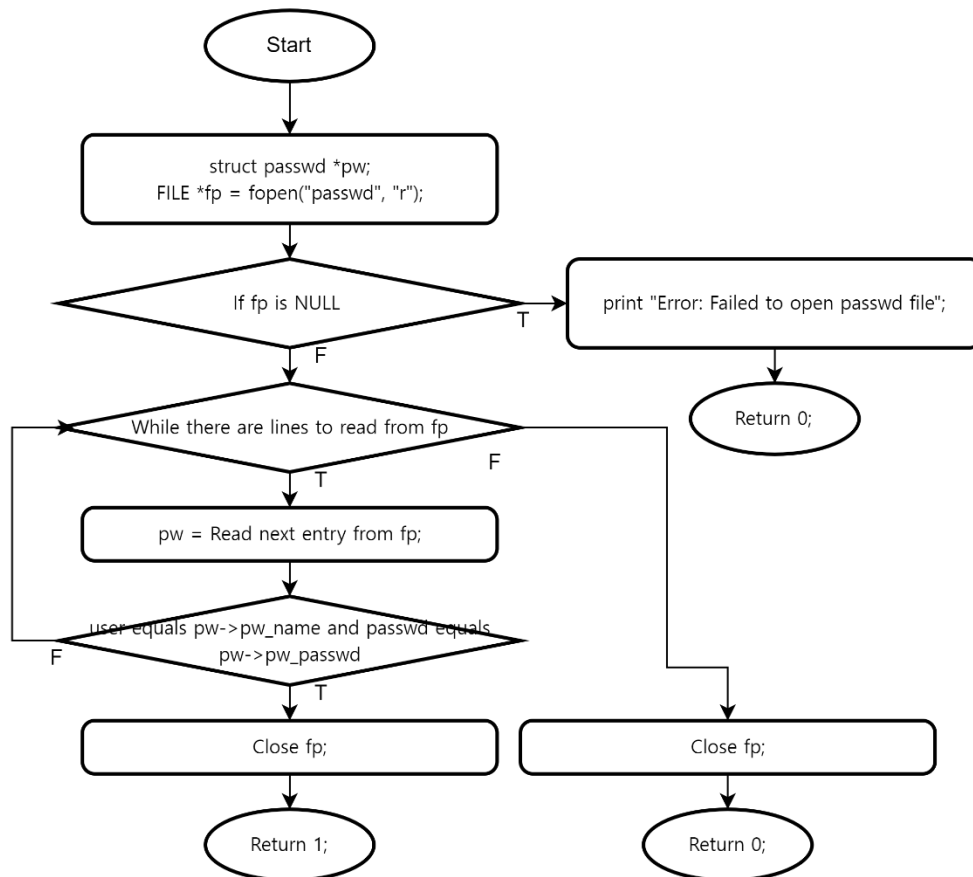
cli.c - log_in()



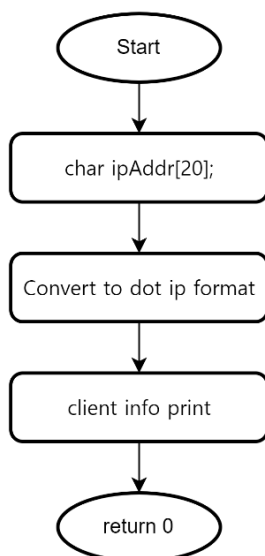
cli.c – main()



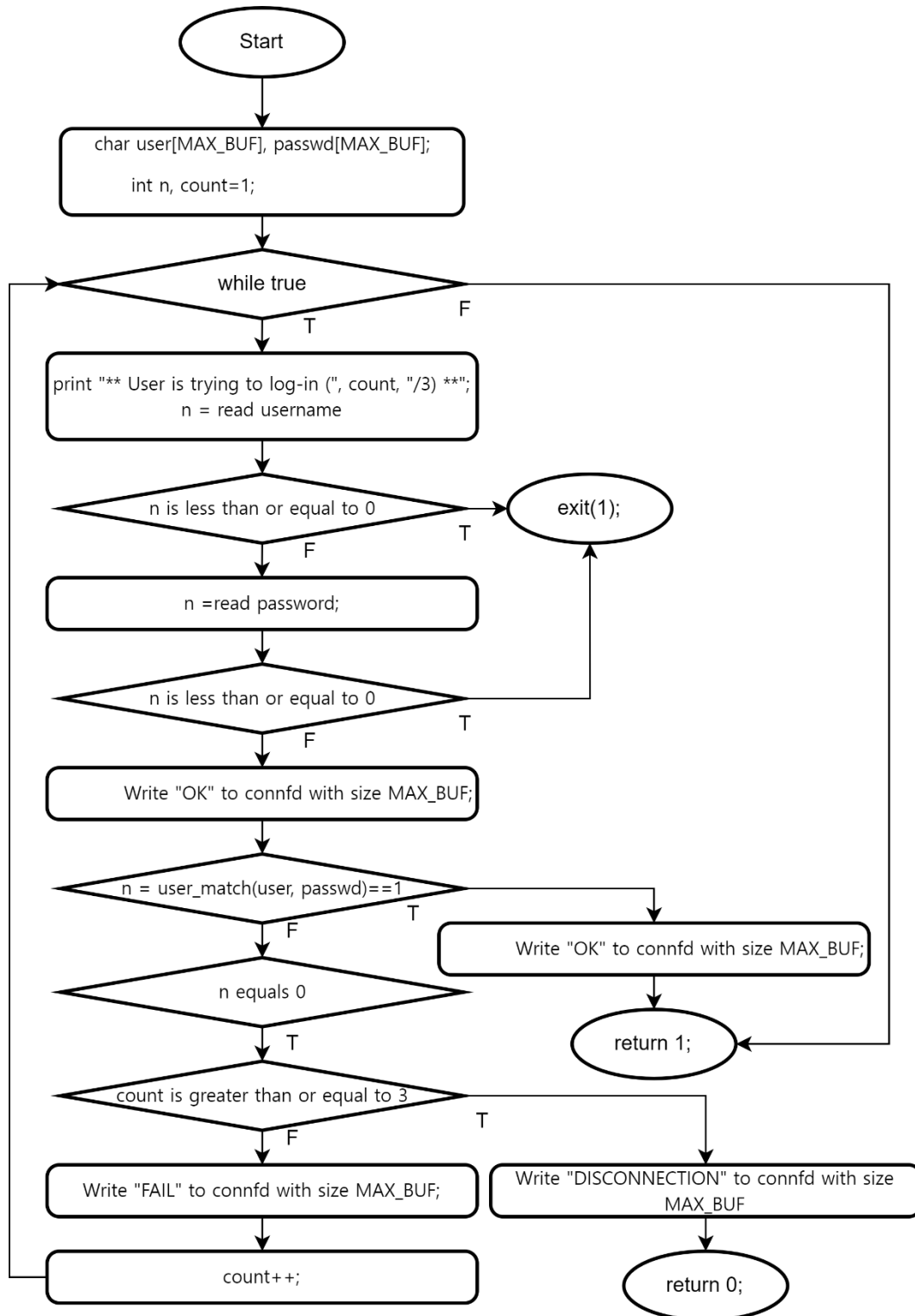
srv.c – user_match



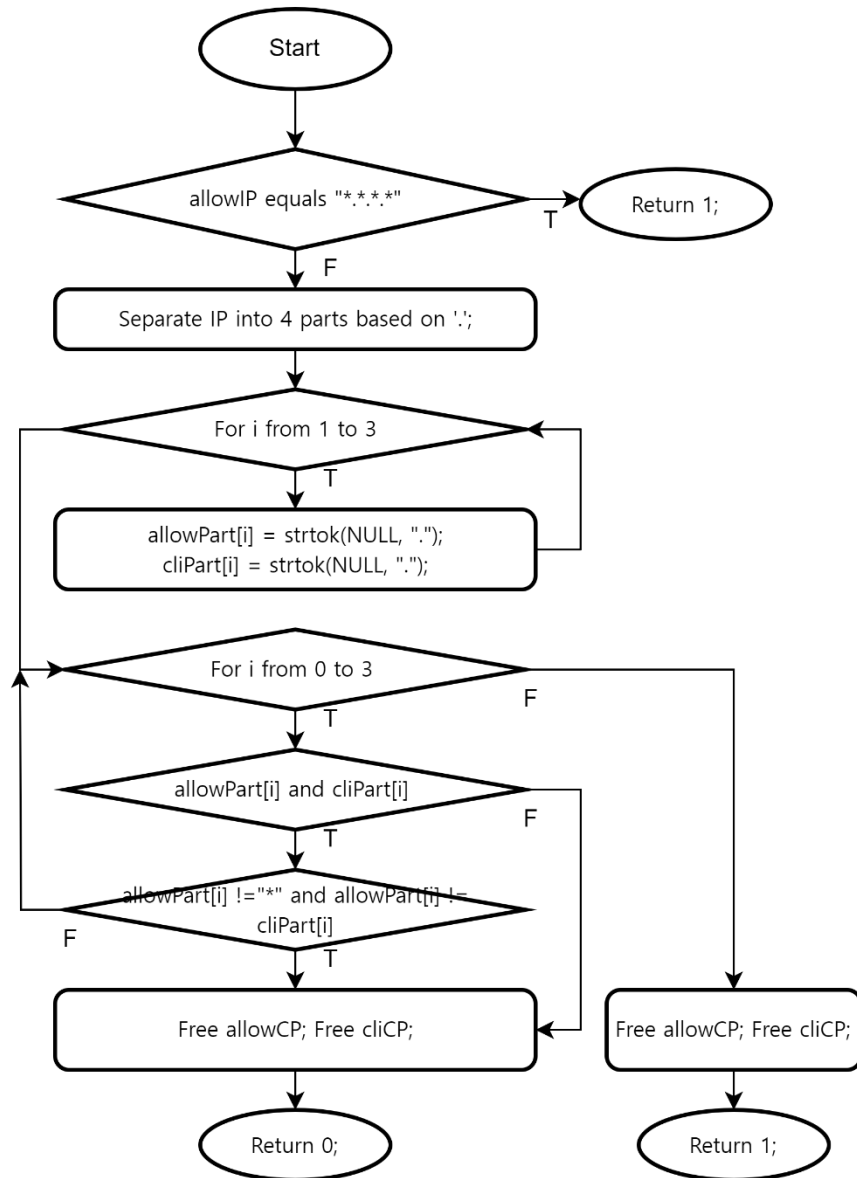
srv.c – client_info()



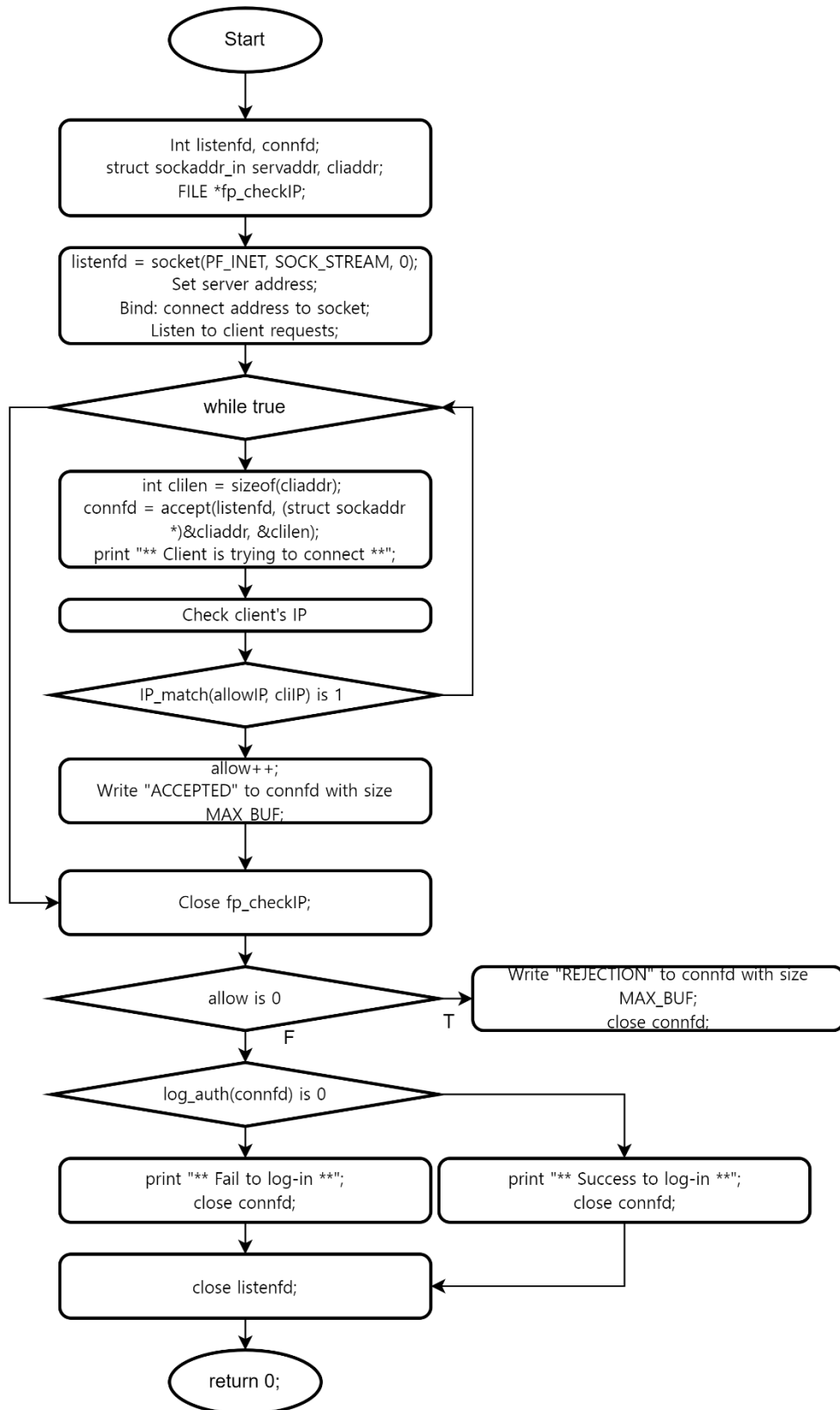
srv.c – log_auth()



srv.c – IP_match



srv.c – main()



Pseudo code

cli.c

```
void log_in(sockfd) {  
  
    int n; char user[MAX_BUF], buf[MAX_BUF];  
  
    // Check if the IP is acceptable  
  
    Check IP result;  
  
    If buf equals "REJECTION" then:  
  
        print "*** Connection refused ***";  
  
        Close sockfd;  
  
        Exit program;  
  
    Else:          // buf is "ACCEPTED", receive username and password  
  
        print "*** It is connected to Server ***";  
  
        While True:  
  
            Get user name;  
  
            Get password;  
  
            pass username and password to server;  
  
            Read results checked by the server;  
  
            If buf equals "OK" then: // Server finds username and password  
  
                n = read from sockfd into buf with size MAX_BUF;  
  
                If buf equals "OK" then: // login success  
  
                    print "*** User '", user, "' logged in ***";  
  
                    Return;  
  
                Else if buf equals "FAIL" then: // login fail  
  
                    print "*** Log-in failed ***";  
  
                Else: // buf is "DISCONNECTION", three times fail  
  
                    prnit "*** Connection closed ***";  
  
                    Close sockfd;  
  
                    Exit program;  
  
}
```

```

int main(int argc, char *argv[]) {

    int sockfd, n, p_pid;

    struct sockaddr_in servaddr;

    char *haddr; //IP address

    haddr = argv[1];

    int PORTNO = atoi(argv[2]); //port

    Creat socket;

    Set server address;

    Connect to server;

    log_in(sockfd);

    Close sockfd;

    Return 0;

}

```

srv.c

```

int user_match(user, passwd) {

    struct passwd *pw;

    FILE *fp = fopen("passwd", "r");

    If fp is NULL then:

        print "Error: Failed to open passwd file";

        Return 0;

    // Check login information

    While there are lines to read from fp:

        pw = Read next entry from fp;

        // Check user name and password

        If user equals pw->pw_name and passwd equals pw->pw_passwd then:

            Close fp;

            Return 1; // Authentication successful

        Close fp;

        Return 0; // Authentication failed

}

```

```

int log_auth(int connfd) {

    char user[MAX_BUF], passwd[MAX_BUF];

    int n, count=1;

    While True {

        print "*** User is trying to log-in (" , count, "/3) ***";

        n = read username;

        If n is less than or equal to 0 then:

            Exit program;

        n =read password;

        If n is less than or equal to 0 then:

            Exit program;

        passwd[n] = End of String;

        Write "OK" to connfd with size MAX_BUF;

        // Check whether the saved name and password match

        n = user_match(user, passwd);

        If n equals 1 then: // success

            Write "OK" to connfd with size MAX_BUF;

            Return 1;

        Else If n equals 0 then: // fail

            If count is greater than or equal to 3 then: // 3 times fail

                Write "DISCONNECTION" to connfd with size MAX_BUF;

                Return 0;

            print "*** Log-in failed ***";

            Write "FAIL" to connfd with size MAX_BUF;

            count++;

            Continue;

    }

    Return 1;

}

```

```

int IP_match(allowIP, cliIP) {

    If allowIP equals "*.*.*.*" then:

        Return 1;

    // Separate IP into 4 parts based on '.'

    char *allowPart[4], *cliPart[4];

    // Copy original IPs

    char *allowCP = strdup(allowIP);

    char *cliCP = strdup(cliIP);

    // Split the IPs into parts

    allowPart[0] = strtok(allowCP, ".");

    cliPart[0] = strtok(cliCP, ".");

    For i from 1 to 3 do:

        allowPart[i] = strtok(NULL, ".");

        cliPart[i] = strtok(NULL, ".");

    // Match check for each part

    For i from 0 to 3 do:

        If allowPart[i] and cliPart[i] are both not null then:

            If allowPart[i] != "*" and allowPart[i] != cliPart[i] then:

                Free allowCP; Free cliCP; Return 0;

            Else: // Fail if invalid

                Free allowCP; Free cliCP; Return 0

        Free allowCP; Free cliCP; Return 1;

}

```

```

int client_info(struct sockaddr_in *client_addr) {

    char ipAddr[20];

    convert to dot ip format;

    print IP address;

    print port;

    return 0;

}

```

```

int main(int argc, char *argv[]) {

    Int listenfd, connfd;

    struct sockaddr_in servaddr, cliaddr;

    FILE *fp_checkIP;

    listenfd = socket(PF_INET, SOCK_STREAM, 0);

    Set server address;

    Bind: connect address to socket;

    Listen to client requests;

    // Connect to client, read and execute

    While True:

        int clilen = sizeof(cliaddr);

        connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen);

        print "*** Client is trying to connect ***";

        // Check client's IP

        char cliIP[INET_ADDRSTRLEN];

        Convert cliaddr.sin_addr to text and store in cliP;

        fp_checkIP = open file "access.txt" for reading;

        If fp_checkIP is NULL then:

            print"Error: Failed to open 'access.txt' file";

            Exit program;

        int allow = 0; //1 if client IP is allowed

        char allowIP[INET_ADDRSTRLEN]; //accessible IP

        // Check the file by reading it line by line

        While read a line from fp_checkIP into allowIP is not NULL:

            Remove newline character from allowIP;

            If IP_match(allowIP, cliIP) is 1 then: // Connect if the IP is accessible

                allow++;

                print "*** Client is connected ***";

                Write "ACCEPTED" to connfd with size MAX_BUF;

                Break;

```

```
Close fp_checkIP;

If allow is 0 then: // Inaccessible IP, connection terminated

    Write "REJECTION" to connfd with size MAX_BUF;

    print "*** It is NOT authenticated client: ", cliIP, " ***";

    Close connfd;

    Continue;

// Check login information

If log_auth(connfd) is 0 then: // if 3 times fail (ok: 1, fail: 0)

    print "*** Fail to log-in ***";

    Close connfd;

    Continue;

print "*** Success to log-in ***";

Close connfd;

Close listenfd;

Return 0;

}
```

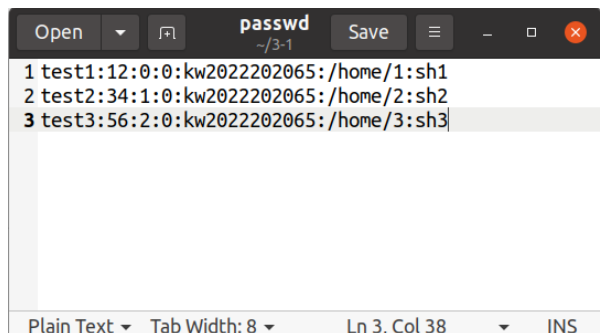
결과화면

1. 접속이 불가능한 IP 를 가진 client 가 접속할 경우

아래와 같이 접속 가능한 IP 주소가 없을 때, 클라이언트에서 연결을 요청하자마자 서버에서 해당 IP 정보를 출력 후 연결을 거부하며 끝난다.



```
1
```



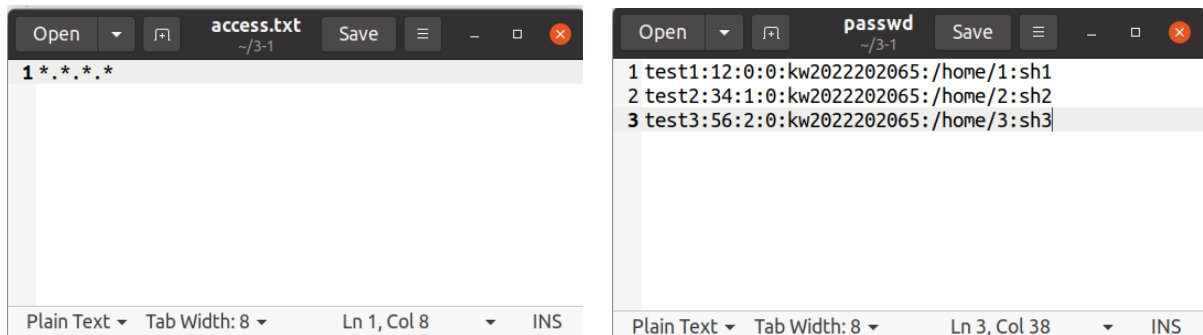
```
1 test1:12:0:0:kw2022202065:/home/1:sh1
2 test2:34:1:0:kw2022202065:/home/2:sh2
3 test3:56:2:0:kw2022202065:/home/3:sh3
```

```
kw2022202065@ubuntu:~/3-1$ ./cli 127.0.0.1 3000
** Connection refused **
kw2022202065@ubuntu:~/3-1$
```

```
kw2022202065@ubuntu:~/3-1$ ./srv 3000
** Client is trying to connect **
- IP: 127.0.0.1
- port: 50264
** It is NOT authenticated client: 127.0.0.1 **
```

2. 성공적으로 로그인을 마친 경우

모든 IP 를 허용하도록 wildcard 로 되어 있고, 저장된 이름과 비밀번호대로 입력하면 아래와 같이 한번에 로그인을 성공할 수 있다. 이때 비밀번호는 getpass()로 받기 때문에 입력창에서는 가려진다.



첫번째 라인에 저장되어 있는 test1:12 로 입력했을 때의 결과이다.

```
kw2022202065@ubuntu:~/3-1$ ./cli 127.0.0.1 3000
** It is connected to Server **
Input ID: test1
Input Password:
** User 'test1' logged in **
kw2022202065@ubuntu:~/3-1$
```

```
kw2022202065@ubuntu:~/3-1$ ./srv 3000
** Client is trying to connect **
- IP: 127.0.0.1
- port: 39970
** Client is connected **
** User is trying to log-in (1/3) **
** Success to log-in **
```


1, 2 번의 시도에서는 실패하고 마지막 3 번째에 성공했을 때의 결과화면이다.
카운트를 세면서 실패로 나타나다가 마지막에만 성공하고 종료된다.

```
kw2022202065@ubuntu:~/3-1$ ./cli 127.0.0.1 3000
** It is connected to Server **
Input ID: test
Input Password:
** Log-in failed **
Input ID: test
Input Password:
** Log-in failed **
Input ID: test1
Input Password:
** User 'test1' logged in **
kw2022202065@ubuntu:~/3-1$
```

```
kw2022202065@ubuntu:~/3-1$ ./srv 3000
** Client is trying to connect **
- IP: 127.0.0.1
- port: 33222
** Client is connected **
** User is trying to log-in (1/3) **
** Log-in failed **
** User is trying to log-in (2/3) **
** Log-in failed **
** User is trying to log-in (3/3) **
** Success to log-in **
```

3. 로그인 세 번 시도했지만 모두 실패한 경우

passwd 파일에 저장되어 있는 정보와 다르게 입력한 경우 전부 실패로 출력된다. 카운트를 세다가 마지막 세번째 시도까지 실패할 시, 연결을 종료한다.

```
kw2022202065@ubuntu:~/3-1$ ./cli 127.0.0.1 4000
** It is connected to Server **
Input ID: test
Input Password:
** Log-in failed **
Input ID: test1
Input Password:
** Log-in failed **
Input ID: test2
Input Password:
** Connection closed **
kw2022202065@ubuntu:~/3-1$
```

```
kw2022202065@ubuntu:~/3-1$ ./srv 4000
** Client is trying to connect **
- IP: 127.0.0.1
- port: 35026
** Client is connected **
** User is trying to log-in (1/3) **
** Log-in failed **
** User is trying to log-in (2/3) **
** Log-in failed **
** User is trying to log-in (3/3) **
** Fail to log-in **
```

고찰

처음에 코드를 작성하고 실행하였을 때, IP 검사가 제대로 이루어지지 않아서 와일드카드를 입력했음에도 불구하고 바로 연결이 종료되었다. 원래는 두 개의 IP를 비교할 때 포인터를 이용하여 한 문자씩 비교하는 방식으로 구현하였다가, wildcard가 있으면 한 문자씩 비교하기 어렵다는 걸 깨달았다. 그래서 IP address를 '.'을 기준으로 4파트로 나누어서 각 배열에 저장한 다음 배열마다 strcmp()로 비교하였다. 그렇게 하니 wildcard가 있어도 정상적으로 수행될 수 있었다.

연결에 성공한 후, 클라이언트에서 input 화면이 바로 뜨지 않았다가 어떠한 문자를 입력한 다음에 버퍼가 꼬인 것처럼 다음 input과 연결 성공 메시지가 한번에 뜨는 상황이 발생하였었다. 나머지 코드를 주석처리하고 하나씩 다시 확인해 보니, 'input username:'은 printf()로 받았다가 그 내용은 read로 읽고 write로 서버에 보낸 다음 다시 printf()로 'input password:'를 출력하는 과정에서 순서 오류가 난 것 같았다. 따라서 read로 읽지 않고 scanf()로 받은 다음 printf()로 하니 차례대로 입력창에 출력될 수 있었다.

서버에 로그인 정보를 보내는 과정에서 'Segmentation fault' 오류가 계속 발생하였는데, 이는 포인터를 선언할 때 바로 초기화를 해주지 않아서 발생한 오류였다. 선언과 동시에 초기화를 하도록 코드를 수정하였더니 정상적으로 처리될 수 있었다.

Reference

강의자료 참고